

CON-FOLD

Explainable Machine Learning with Confidence

LACHLAN MCGINNESS

School of Computer Science, ANU and CSIRO/Data61, Canberra, Australia
(e-mail: lachlan.mcginness@anu.edu.au)

PETER BAUMGARTNER

CSIRO/Data61 and School of Computer Science, ANU, Canberra, Australia
(e-mail: peter.baumgartner@data61.csiro.au)

submitted 16 August 2024; accepted 13 September 2024

Abstract

FOLD-RM is an explainable machine learning classification algorithm that uses training data to create a set of classification rules. In this paper, we introduce CON-FOLD which extends FOLD-RM in several ways. CON-FOLD assigns probability-based confidence scores to rules learned for a classification task. This allows users to know how confident they should be in a prediction made by the model. We present a confidence-based pruning algorithm that uses the unique structure of FOLD-RM rules to efficiently prune rules and prevent overfitting. Furthermore, CON-FOLD enables the user to provide preexisting knowledge in the form of logic program rules that are either (fixed) background knowledge or (modifiable) initial rule candidates. The paper describes our method in detail and reports on practical experiments. We demonstrate the performance of the algorithm on benchmark datasets from the UCI Machine Learning Repository. For that, we introduce a new metric, Inverse Brier Score, to evaluate the accuracy of the produced confidence scores. Finally, we apply this extension to a real-world example that requires explainability: marking of student responses to a short answer question from the Australian Physics Olympiad.

KEYWORDS: logic programming methodology and applications, inductive logic programming and multi-relational data mining

1 Introduction

Machine learning (ML) has been shown to be incredibly successful at learning patterns from data to solve problems and automate tasks. However, it is often difficult to interpret and explain the results obtained from ML models. Decision trees are one of the few ML methods that offer transparency with regard to how decisions are made. They allow users to follow a set of rules to determine what the outcome of a task (say, classification) should be. The difficulty with this approach is finding an algorithm that is able to construct a reliable set of decision trees.

One approach of generating a set of rules equivalent to a decision tree is the First Order Learner of Default (FOLD) approach introduced by (Shakerin *et al.* 2017). To improve a model's ability to handle exceptions in rule sets, Shakerin, Wang, Gupta, and others introduced and refined an explainable ML algorithm called First Order Learner of Default (FOLD) (Shakerin *et al.* 2017; Wang 2022; Wang and Gupta 2022; Wang *et al.* 2022; Wang and Gupta 2023; Padalkar *et al.* 2024), which learns non-monotonic stratified logic programs (Quinlan, 1990b). The FOLD algorithm is capable of handling numerical data (FOLD-R) (Shakerin *et al.* 2017), multi-class classification (FOLD-RM) (Wang *et al.* 2022), and image inputs (NeSyFOLD) (Padalkar *et al.* 2024). FOLD-SE uses Gini Impurity instead of information gain in order to obtain a more concise set of rules (Wang and Gupta 2023). Thanks to these improvements, variants of the FOLD algorithm are now competitive with state-of-the-art ML techniques such as XGBoost and RIPPER in some domains (Wang and Gupta 2022, 2023).

The rules produced by the FOLD algorithm are highly interpretable; however, they can be misleading. As an example let's consider the popular Titanic dataset (Kaggle 2012), where passengers are classified into two categories: perished or survived. One rule from the FOLD algorithm might say that a passenger survives if they are female and do not have a third-class ticket. When given a new set of data and this rule, a user might be unpleasantly surprised to find that such a passenger perished. This is because the rule can have the appearance of being definitive. In reality, this could be a good rule that is correct 99% of the time. In order for a FOLD model to be more understandable and trustworthy for users, a confidence value could be provided. This would provide the user with a measure of the certainty of the rule and would make it clear to the user that not all women with second-class tickets survive.

In this paper, we introduce Confidence-FOLD (CON-FOLD), an extension of the FOLD-RM algorithm. CON-FOLD provides confidence values for each rule generated by the FOLD algorithm so users know how confident they should be for each rule in the model. In addition, we present a pruning algorithm that makes use of these confidence values. These techniques applied to the Titanic example yield the following rules and confidences¹:

```

survived(X, false) :- rule1(X).           %confidence: 0.9
survived(X, true) :- rule2(X), not rule1(X). %confidence: 0.97
rule1(X) :- not sex(X, female).
rule2(X) :- sex(X, female).

```

We also provide a metric, Inverse Brier Score, which can be used to evaluate probabilistic or confidence-based predictions that maintain compatibility with the traditional metric of accuracy. We provide the capability to introduce modifiable initial knowledge into a FOLD model. Finally, we demonstrate the effectiveness of adding background knowledge in the marking of student responses to physics questions. Note that we choose to focus on multi-class classification tasks in our experiments; however, CON-FOLD can also be applied to binary classification.

¹ For simplicity of demonstration, the rules were obtained from the Titanic *test* dataset. We used an improvement threshold of 0.1 (Section 4.1) and a confidence threshold of 0.5 (Section 4.2).

2 Formal framework and background

We work with the usual logic programming terminology. A (*logic program*) rule is of the form

$$h :- l_1, \dots, l_k, \text{not } e_1, \dots, \text{not } e_n . \tag{1}$$

where the h, l_1, \dots, l_k , and e_1, \dots, e_n all are atoms, for some $k, n \geq 0$. A *program* is a finite set of rules. We adopt the formal learning framework described for the FOLD-RM algorithm. In this, two-ary predicate symbols are used for representing feature values, which can be categorical or numeric. Example feature atoms are `name(i1,sam)` and `age(i1,30)` of an individual `i1`. Auxiliary predicates and Prolog-like built-in predicates can be used as well. Rules always pertain to one single individual and its features as in this example:

$$\text{female}(X) :- \text{rule1}(X) . \tag{1}$$

$$\text{rule1}(X) :- \text{age}(X,A), A > 16, \text{not } \text{ab}(X) . \tag{2}$$

$$\text{ab}(X) :- \text{name}(X,\text{sam}), \text{not } \text{fav_color}(X,\text{purple}) . \tag{3}$$

These rules for a target relation `female` could have been learned from training data where all individuals older than 16 are female, except those named `sam` and whose favorite color is not `purple`. Below, we use the letter r to refer to the rule for the target relation, $r = \text{female}$, (1) in this example, and the letter R for the set of auxiliary rules, $R = \{(2), (3)\}$, that are needed to define the predicates in r .

The learning task in general is defined in Wang *et al.* (2022). The learning algorithm takes as input two disjoint sets $X = X_p \uplus X_n$ of *positive and negative training examples*, respectively. It assumes that the training set distribution is approximately the same as the test set distribution.

For any $d \in X$, let $features(d)$ denote d 's features as a set of atoms over some a priori fixed Skolem constant, say, c , for example: $features(d) = \{\text{age}(c, 18), \text{name}(c, \text{adam}), \text{fav_color}(c, \text{red})\}$. The learning algorithm below checks if a current set $R \cup \{r\}$ puts an example d into the target class. Letting ℓ denote the target class as an atom, for example $\ell = \text{female}(c)$, this is accomplished via an entailment check $R \cup \{r\} \cup features(d) \models \ell$.

Because all learned programs are stratified, we adopt the standard semantics for that case, the perfect model semantics (layered bottom-up fixpoint computation), which is also reflected in the FOLD-RM algorithm. We emphasize that default negation causes no problems in calculating the confidence scores of a rule.

The confidence scores are attached only to the top-level rules r , never to the rules R referred to under default negation. That is, having to deal with confidence scores in a negated context will never be necessary. We will describe the rationale behind this design decision shortly below.

The Boolean learning task is to determine a stratified program P such that

$$P \cup features(d) \models \ell \text{ for all } d \in X_p \quad \text{and} \quad P \cup features(d) \not\models \ell \text{ for all } d \in X_n .$$

The multi-class learning problem is a generalization to a finite set of classes. The training data X then is comprised of atoms indicating the target class, for each data point,

for example, `survived(c,false)` or `survived(c,true)` in the Titanic example. Conversely, by splitting multi-class learning problems can be treated as a sequence of Boolean learning problems.

The basis of CON-FOLD is the FOLD-RM algorithm (Wang *et al.* 2022). FOLD-RM simplifies a multi-class classification task into a series of Boolean classification task. It first chooses the class with the most examples and sets data which correspond to this class as positive and all other classes as negative. It then generates a rule that uses input features to maximize the information gain. This process is repeated on a subset of the training examples by eliminating those that are correctly classified with the new rule. The process stops when all examples are classified.

3 Related work

Confidence scores have been introduced in decision tree learning for assessing the admissibility of a pruning step (Quinlan 1990a). In essence, decision tree pruning removes a sub-tree if the classification accuracy of the resulting tree does not fall below a certain threshold, for example, in terms of standard errors, and possibly corrected for small domain sizes. Decision trees can be expressed as sets of production rules (Quinlan 1987), one production rule for each branch in a decision tree. The production rules can again be simplified using scoring functions (Quinlan 1987).

The FOLD family of algorithms learns rules with exceptions by means of default negation (negation-as-failure). The rules defining the exceptions can have exceptions themselves. This sets FOLD apart from the production systems learned by decision tree classifiers, which do not take advantage of default negation. This may lead to more complex rule sets. Indeed, (Wang *et al.* 2022) observe that “For most datasets we experimented with, the number of leaf nodes in the trained C4.5 decision tree is much more than the number of rules that FOLD-R++/FOLD-RM generate. The FOLD-RM algorithm outperforms the above methods in efficiency and scalability due to (i) its use of learning defaults, exceptions to defaults, exceptions to exceptions, and so on, (ii) its top-down nature, and (iii) its use of improved method (prefix sum) for heuristic calculation.” We do feel, however, that these experimental results could be completed from a more conceptual point of view. This is beyond the scope of this paper and left as future work.

Scoring functions have been used in many rule-learning systems. The common idea is to allow accuracy to degrade within given thresholds for the benefit of simpler rules. See Law *et al.* (2020) for a discussion of the more recent ILASP3 system and the references therein. Hence, we do not claim the originality of using scoring systems for rule learning. We see our main contribution differently and not in competition with other systems. Indeed, one of the main goals of this paper is to equip an *existing* technique that has been shown to work well – FOLD-RM – with confidence scores. With our algorithm design and experimental evaluation we show that this goal can be achieved in an “almost modular” way. Moreover, as our extension requires only minimal changes to the base algorithm, we expect that our method is transferrable to other rule-learning algorithms.

4 The CON-FOLD algorithm and confidence scores

The CON-FOLD algorithm assigns each rule a confidence score as it is created. For easy interpretability, confidence scores should be equal to probability values (p-values from a binomial distribution) in the case of large amounts of data. However, p-values would be a very poor approximation in the case of small amounts of training data; if a rule covered only one training example, it would receive a confidence value of 1 (100%). There are many techniques for estimating p-values from a sample, we chose the center of the Wilson Score Interval (Wilson, 1927) given by Eq. (2). The Wilson Score Interval adjusts for the asymmetry in the binomial distribution which is particularly pronounced in the case of extreme probabilities and small sample sizes. It is less prone to producing misleading results in these situations compared with the normal approximation method, thus making it more trustworthy for users (Agresti and Coull 1998).

$$p = \frac{n_p + \frac{1}{2}Z^2}{n + Z^2}, \tag{2}$$

where p is the confidence score, n_p is the number of training examples corresponding to the target class covered by the rule, n is the number of training examples covered by the rule corresponding to all classes, and Z is the standard normal interval half width; by default, we use $Z = 3$.

Theorem 4.1.

In the limit where there is a large amount of data classified by a rule ($n \rightarrow \infty$), the confidence score approaches the true probability of the sample being from the target class.

Proof

The true probability that a randomly selected example that follows the provided rule is from the target class is $p_r = \frac{n_p}{n}$. As n increases, the law of large numbers states that the portion of examples which belong to the target class becomes $\lim_{n \rightarrow \infty} n_p = p_r n$. Also, as $n \rightarrow \infty$, the relative contribution of Z^2 terms becomes negligible. Therefore:

$$\lim_{n \rightarrow \infty} p = \lim_{n \rightarrow \infty} \frac{n_p + \frac{1}{2}Z^2}{n + Z^2} = \lim_{n \rightarrow \infty} \frac{np_r}{n} = p_r$$

□

Once rules have the associated confidence scores, they are expressed as follows:

$$p :: h :- l_1, \dots, l_k, \text{not } e_1, \dots, \text{not } e_n . \tag{3}$$

where p is the confidence score. The format of confidence score annotations is directly supported by probabilistic logic programming systems such as Problog (De Raedt *et al.* 2007) and Fusemate (Baumgartner and Tartaglia 2023). In this paper, we do not explore this possibility and just let the confidence scores allow the user to know the reliability of a prediction made by a rule in the logic program.





The CON-FOLD algorithm (Algorithm 1) closely follows the presentation of FOLD-RM in Wang *et al.* (2022); see there for definitions of SPLIT_BY_LITERAL, LEARN_RULE (slightly adapted) and MOST. On line 11, $conf(P, X_p, X_n)$ computes the Wilson score

Algorithm 1 CON-FOLDCON-FOLD(X, t)**Input:** X training examples, t threshold**Output:** program P for classifying X

```

1   $P \leftarrow \emptyset$  // Result
2  while  $X \neq \emptyset$  do
3     $l \leftarrow \text{MOST}(X)$ 
4     $X_p, X_n \leftarrow \text{SPLIT\_BY\_LITERAL}(X, l)$ 
5     $r, R \leftarrow \text{LEARN\_RULE}(l, X_p, X_n)$ 
6     $R \leftarrow \text{EVALUATE\_EXCEPTIONS}(r, R, X_p, X_n, t)$ 
7     $X_{fn} \leftarrow \{d \in X_p \mid R \cup \{r\} \cup \text{features}(d) \neq l\}$ 
8    if  $|X_{fn}| = |X_p|$  then break // End if rule does not correctly classify any examples
9     $X_{tn} \leftarrow \{d \in X_n \mid R \cup \{r\} \cup \text{features}(d) \neq l\}$ 
10    $X \leftarrow X_{fn} \cup X_{tn}$ 
11    $c \leftarrow \text{conf}(R \cup \{r\}, X_p, X_n)$ 
12    $P \leftarrow P \cup R \cup \{c :: r\}$ 
13 return  $P$ 

```

Bird	Owl	Eagle	Flamingo	Humming Bird
				
Size	Large	Large	Large	Small
Beak Shape	Round	Round	Round	Long
Predator	1	1	0	0

```

predator(X,1) :- rule1(X).           rule1(X) :- large(X).
predator(X,0) :- rule2(X), not rule1(X).  rule2(X) :- ? Should this rule consider the Flamingo?

```

Fig. 1. This toy example illustrates the difference between the FOLD-RM and CON-FOLD core algorithms. Both produce rules of the form shown. CON-FOLD would not consider the Flamingo as part of the data to fit when generating rule 2. FOLD-RM would consider the Flamingo. Note that in many cases, both algorithms would generate an abnormal rule $\text{ab}(X) :- \text{flamingo}(X)$, preventing the Flamingo from being covered by the first rule. In this case, both FOLD-RM and CON-FOLD would include the Flamingo. When harsh pruning occurs and there are few abnormal rules, this subtle change becomes noticeable.

p as in (2), letting

$$n_p = |\{d \in X_p \mid P \cup \text{features}(d) \models \ell\}|,$$

$$n = n_p + |\{d \in X_n \mid P \cup \text{features}(d) \models \ell\}|$$

and ℓ the target class as an atom. Note that line 6 is only used if pruning. Other than this, the only difference between CON-FOLD and FOLD-RM are lines 9 and 10. In our terminology, FOLD-RM includes in the updated set X the full set X_n instead of X_{tn} . The consequence of this difference is highlighted in Figure 1.

Theorem 4.2.

CON-FOLD algorithm always terminates on any set of finite examples.

Proof

Each pass through the while loop produces a rule that will either successfully classify at least one example or not. If no examples are successfully classified, the algorithm terminates immediately (lines 8–9). Otherwise, the examples classified are removed from the set (line 11) strictly decreasing its size. Since the set is finite and each cycle removes at least one element, it will become empty eventually, and the loop will terminate on that condition. \square

FOLD-RM has a complexity of $\mathcal{O}(NM^3)$ where N is the number of features and M is the number of examples (Wang *et al.* 2022). In the worst case, each literal only covers one example and requires an additional $M - 1$ literals to exclude the remaining data. Then the pruning algorithm can be called once per rule for each target class, which in the worst case is M times. The complexity of the pruning algorithm is below.

Once confidence values have been assigned to each rule, it is possible to use these values to allow for pruning and prevent overfitting. In the following, we introduce two such pruning methods: *improvement threshold* and *confidence threshold* pruning.

4.1 Improvement threshold pruning

Improvement threshold pruning is designed to stop rules from overfitting data by becoming unnecessarily “deep.” The rule structure in FOLD allows for exceptions, for exceptions to exceptions, and then for exceptions to these exceptions and so on. This may overfit the model to noise in the data very easily. We will refer to any exception to an exception at any depth as a *sub-exception*.

In order to avoid this overfitting, each time a rule is added to the model, each exception to the rule is temporarily removed, and a new confidence score is calculated. If this changes the confidence by less than the improvement threshold, then this exception is removed. If an exception is kept, then this process is applied to each sub-exception. This process is repeated until each exception and sub-exception has been checked.

The details are formalized in the pseudocode in Algorithm 2. In there, $remove_rule(R, r)$ removes the rule r from the set R of rules and removes the possibly default-negated atom with the head of r from the bodies of all rules in R .

Theorem 4.3.

The Evaluate Exceptions algorithm always terminates on any set of finite set of rules, exceptions, and data points.

Proof

On any given rule, each exception is checked once, and the algorithm terminates when there are no more exceptions to be checked. The depth of recursions and therefore the number of times that EVALUATE_EXCEPTIONS is called recursively is bound by the number of exceptions. \square

To determine the complexity of this pruning algorithm, let R be the total number of exceptions and sub-exceptions and $M = |X_p| + |X_n|$ be the number of examples.

Algorithm 2 Evaluate ExceptionsEVALUATE_EXCEPTIONS(r, R, X_p, X_n, t)**Input:** r - rule of form $h_0 :- l_{r_1}, \vec{e}_0$ where l_{r_1} is the head of the rule $r_1 \in R$ R - set of auxiliary rules for r , each of form $h :- \vec{l}, \vec{e}$ where $\vec{e} = \text{not } e_1, \dots, \text{not } e_n$
where each e_i is the head of a rule $r_{e_i} \in R$ X_p - positive examples, X_n - negative examples, t - threshold**Output:** Pruned version of auxiliary rules R 1 **return** EV_EX_LOOP(r, R, r_1, X_p, X_n, t)EV_EX_LOOP(r, R, r', X_p, X_n, t)**Input:** r, R, X_p, X_n, t as in EVALUATE_EXCEPTIONS r' - rule in R of form $h :- \vec{l}, \vec{e}$ where $\vec{e} = \text{not } e_1, \dots, \text{not } e_m$
and each e_i is the head of a rule $r_{e_i} \in R$ **Output:** Pruned version of auxiliary rules R

```

1   $R' \leftarrow R$ 
2  for  $e_i \in \vec{e}$  do
3     $R_t \leftarrow \text{remove\_rule}(R', r_{e_i})$ 
4     $c \leftarrow \text{conf}(r, R', X_p, X_n)$ 
5     $c_t \leftarrow \text{conf}(r, R_t, X_p, X_n)$ 
6    if  $c - c_t < t$  then
7       $R' \leftarrow R_t$  // Tolerable loss of confidence
8    else // Recurse into exceptions for  $e_i$ 
9       $R' \leftarrow \text{ev\_ex\_loop}(r, R', e_i, X_p, X_n, t)$ 
10 return  $R'$ 

```

Therefore, the number of sub-exceptions to any exception is bounded in the worse case by $R - 1$ or $\mathcal{O}(R)$. The calculation of confidence within the loop takes $\mathcal{O}(MR)$ time as it requires comparing each sub-exception to a maximum of M data points. The loop must run once for each exception which in the worst case is $\mathcal{O}(R)$. Therefore the total complexity of running the pruning algorithm is $\mathcal{O}(MR^2)$.

4.2 Confidence threshold pruning

In addition to decreasing the depth of rules, it is also desirable to decrease the number of rules. A *confidence threshold* is used to determine whether a rule is worth keeping in the model or whether it is too uncertain to be useful. If the rule has a confidence value below the confidence threshold, then it is removed. This effectively means that rules could be removed on two grounds:

- There are insufficient examples in the training data to lead to a high confidence value.
- There may be many examples in the training data that match the rule; however, a large fraction of the examples are actually counterexamples that go against the rule.

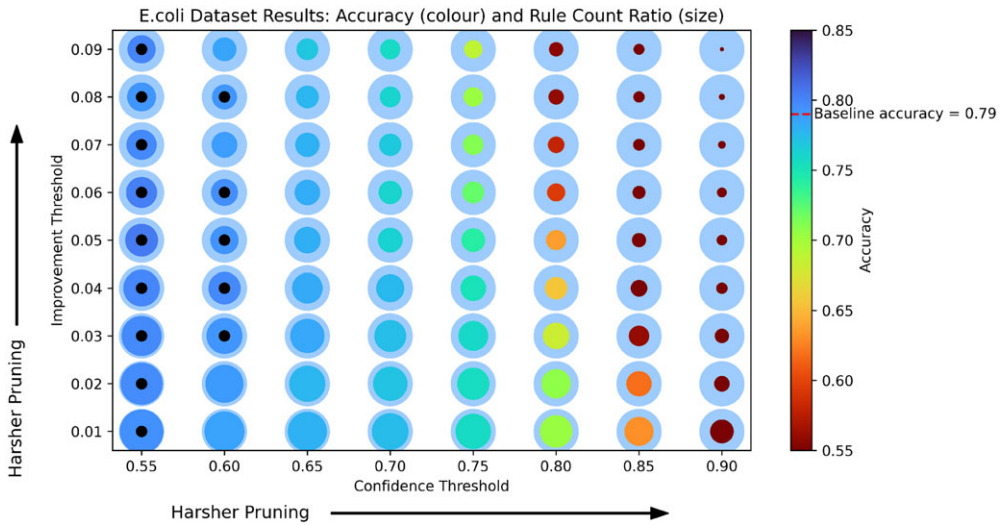


Fig. 2. Scatter plot of the accuracy and number of rules for a ruleset generated by the pruning algorithm with different values of the improvement threshold and the confidence threshold. Each point has two circles. The background circle displays the number of rules and accuracy for no pruning; therefore, all background circles are the same. The front circle displays the rules and accuracy when pruning is applied. The accuracy is indicated by the color shown in the scale bar on the right-hand side. Pruning conditions that are more accurate than the unpruned condition are indicated with a black dot in the center. The number of rules is indicated by the area of the circle (equal amount of ink for number of rules), normalized by the number of rules in the unpruned case. The results shown for both the accuracy and the number of rules are the average of 300 trial runs for each test condition.

The two pruning methods introduced above can greatly simplify a model making it more human interpretable. However, the pruning of rules reduces the model’s ability to fit noise and can lead to an increase in accuracy. However, if rules are pruned too harshly, then the model will be underfit, and performance will decrease. In order to assess these effects we applied CON-FOLD to a sample dataset, the *E.coli* dataset. In our experiments, we varied the values for the two threshold parameters corresponding to the two pruning methods. This allowed us to assess the performance with respect to accuracy and to derive recommendations for parameter settings (see Figure 2). For this dataset, accuracy is highest for pruning with a low confidence threshold and a moderate improvement threshold. If the pruning is too harsh, this leads to a significant decrease in performance.

5 Inverse Brier Score

A standard measure of performance in ML is accuracy which for multi-class tasks is defined by:

$$\text{Accuracy} = \frac{1}{N} \sum_{i=1}^N A(y_i^*, y_i) \tag{4}$$

where $A(y^*, y) = 1$ if $y^* = y$ and 0 otherwise.

When predictions are made with confidence scores, it is possible to use more sophisticated measures of the model's performance. In particular, the model should only be given a small reward if it makes a correct prediction with a low confidence, and the model should be punished when making high confidence predictions that are incorrect. We have three desiderata that for such a scoring system:

1. It is a proper scoring system: the maximum reward can be gained when the probability value given matches the true probability value.
2. The scoring system reduces to accuracy when non-probabilistic predictions are made. This allows for the comparison between probabilistic and non-probabilistic models in terms of performance.
3. The scoring system has an inbuilt mechanism for dealing with no given prediction.

In order to meet all three of these desiderata, we propose a variant of the Brier scoring system and call it Inverse Brier Score (IBS). Brier score (or quadrature score) is often used to evaluate the quality of weather forecasts (Allen *et al.* 2023; Liu *et al.* 2023) and can be obtained with the following formula (Brier 1950):

$$\text{Brier Score} = \frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K (p_{ki} - y_{ik})^2, \quad (5)$$

where i is an index over each data point in a test dataset, k corresponds to a class in the dataset, N is the total number of test examples, and K the total number of classes. This is commonly reduced to the following when only making predictions for one class (Murphy and Epstein 1967):

$$\text{One Class Brier Score} = \frac{1}{N} \sum_{i=1}^N (p_i - y_i)^2 \quad (6)$$

We define the IBS as:

$$\text{IBS} = 1 - \frac{1}{N} \sum_{i=1}^N (p_i - y_i)^2 \quad (7)$$

Theorem 5.1.

Inverse Brier Score is a proper scoring system.

Proof

It known that the Brier Score is a proper scoring system (Murphy and Epstein 1967). Since the propriety of a scoring system is preserved under linear transformations, the IBS is a proper scoring system. \square

Theorem 5.2.

When definite predictions are made ($p = 1$ or $p = 0$), IBS is equivalent to accuracy.

Proof

For non-probabilistic decisions, p_i is replaced by the prediction y^* . Therefore, $(y_i^* - y_i)^2 = 1 - A(y_i^*, y_i)$, and the IBS reduces to the definition of accuracy as follows:

$$1 - \frac{1}{N} \sum_{i=1}^N 1 - A(y_i^*, y_i) = 1 - \frac{N}{N} + \frac{1}{N} \sum_{i=1}^N A(y_i^*, y_i) = \frac{1}{N} \sum_{i=1}^N A(y_i^*, y_i)$$

□

Finally, IBS has a natural way of dealing with no predictions being made. If any class was given a prediction of $p = 0.5$, then IBS would be 0.75 regardless of the class that is chosen. This provides a natural default value if a model refuses to make a prediction due to low confidence, satisfying the third desideratum. Note that this is also important for the FOLD algorithm because it is possible that it will find a test sample, which is significantly different to the training examples and may not match any rules.

Therefore, the IBS satisfies all three desiderata. Both IBS and accuracy are used to evaluate CON-FOLD against other models in Section 7. We note that confidence values are regularly used in weather forecasting where Brier score is used as a metric to evaluate their quality. We use this metric to show that FOLD models with confidence scores obtain higher scores. Although Brier Score does not always align with user’s expectations (Jewson 2004), it is a metric which indicates that models with confidence are more interpretable.

6 Manual addition of rules and physics marking

A key advantage of FOLD over other ML methods is that users are able to incorporate background knowledge about the domain in the form of rules. In addition to fixed background knowledge, we include modifiable initial knowledge. Initial knowledge can be provided with or without confidence values. During the training, CON-FOLD can add confidence values to provided rules, prune exceptions, and even prune these rules if they do not match the dataset.

In order for rules to be added to a FOLD model, they must be admissible rules as defined below, with optional confidence values. Admissible rules obey the following conditions:

- Every head must have a predicate of the value to be decided.
- Each body must consist only of predicates of values corresponding to features.
- Bodies can be Boolean combinations literals.
- The $<$, $>$, \leq , \geq , $=$ and \neq operators are allowed for comparison of numeric variables.
- For categorical variables, only $=$ and \neq are allowed.

Given formulas of this form are translated into logic programs. Stratified default negation is in place to ensure that rule evaluation is done sequentially, mirroring a decision list structure. Therefore, the rules are in a hierarchical structure, and the order in which the initial/background knowledge is added can influence model output in the case of overlapping rule bodies.

The inclusion of background and initial knowledge can be very helpful in cases where only very limited training data is available. An example of such a problem domain is grading a students’ responses to physics problems. Usually, background domain knowledge is easily available in the form of well-defined rules for how responses should be scored.

We evaluated this idea with data from the 2023 Australian Physics Olympiad provided by Australian Science Innovations. The data included 1525 student responses to 38 Australian Physics Olympiad questions and the grades awarded for each of these responses. We chose to mark the first question of the exam because most students attempted it and the answer is simply a number with units and direction. This problem was also favorable because the marking scheme was simple with only three possible marks, 0, 0.5 and 1. Responses to this question were typed, so there was no need for optical character recognition (OCR) to interpret hand-written work. An example of a rule used for marking is:

```
grade(1,X) :- rule1(X).
rule1(X) :- correct_number(X), correct_unit(X).
```

In order to apply the marking scheme, relevant features would need to be extracted from the student responses. Feature extraction is an active area of research in natural language processing (NLP) (Qi 2024). Many approaches focus on extracting information and relationships about entities from large quantities of text and can extract large numbers of features (Carenini *et al.* 2005; Vicient *et al.* 2013). The tools that use term frequency can struggle with short answers (Liu *et al.* 2018) especially if they contain large numbers of symbols and numbers, making them inappropriate for feature extraction for grading physics papers.

In our work, we use SpaCy (Honnibal and Montani 2017) for part-of-speech (POS) tagging to extract noun phrases and numbers to be used as keywords. The presence or absence of these keywords is then one hot encoded for each piece of text to create a set of features. This method alone was not sufficient to extract sufficiently sophisticated features that would allow the marking scheme to be implemented. Therefore, we also used regular expressions to extract the required features for the marking scheme. This required significant customization to match the wide variety of notations used by students.

7 Results

We compare the CON-FOLD algorithm to XGBoost (Chen and Guestrin 2016), a standard ML method, to FOLD-RM, and FOLD-SE, which is currently the state-of-the-art FOLD algorithm. The results can be found in Table 1. The experiments use datasets from the UCI Machine Learning Repository (Kelly *et al.* 2024). The comparison uses 30 repeat trials, where a random 80% of the data is selected for training and the remaining 20% is used for testing.

The hyperparameters for all XGBoost experiments are the defaults in the existing Python implementation.² In all FOLD-RM and CON-FOLD experiments, the ratio is set to 0.5 (default in FOLD-RM).

The hyperparameters for the pruned CON-FOLD algorithm are included with the results. FOLD-SE was not included in the experiments as the implementation is not publicly available, but values from Wang and Gupta (2023) are included. We note that

² `max_depth = 6`, `learning_rate = 0.3`, `n_estimators = 100`, `objective = binary:logistic` and `scale_pos_weight = 1`

Table 1. Accuracy, runtime, and number of rules and predicates for different methods and different benchmarks from the UCI repository. Pruning hyperparameters are provided for the CON-FOLD algorithm. The uncertainty values provided are the standard deviations from 30 trial runs. FOLD-SE results are taken from Wang and Gupta (2023)

Dataset	XGBoost		FOLD-RM						CON-FOLD with pruning				FOLD-SE		
	Time	Acc	Time	Acc	Rules	Preds	t_{con}	t_{imp}	Time	Acc	Rules	Preds	Acc	Rules	Preds
Wine	0.15 s	0.96±0.03	0.02 s	0.94±0.03	7.3±0.6	7.3±0.9	0.55	0.02	0.02 s	0.93±0.03	6.8±0.8	7.0±0.9	0.95	6.5	7.6
E. coli	0.67 s	0.84±0.04	0.04 s	0.79±0.04	39.2±4.6	46±8	0.65	0.08	0.04 s	0.80±0.04	12.2±1.8	16±5	0.80	24	45
Weight Lifting	9.2 s	1.0±0.0	1.7 s	0.999±0.001	14.4±1.2	16.6±1.3	0.90	0.02	1.7 s	0.990±0.005	11.3±0.8	12.4±1.1	1.0	7.0	11
Wall Robot	0.47 s	0.996±0.002	2.0 s	0.993±0.003	30.1±2.4	41±4	0.65	0.01	2.6 s	0.988±0.003	20.0±1.8	25±3	0.99	7.1	16
Page Blocks	0.75 s	0.972±0.004	0.93 s	0.967±0.005	65.1±9.5	112±26	0.70	0.09	1.9 s	0.94±0.01	4.1±1.2	4.4±2.0	0.96	8.5	15
Nursery	0.68 s	0.999±0.001	0.90 s	0.96±0.01	71.4±8.0	26±3	0.55	0.04	1.9 s	0.92±0.01	23±3	15.2±2.5	0.92	18	40
Dry Bean	1.3 s	0.928±0.004	9.6 s	0.911±0.005	186±16	303±37	0.65	0.01	14.3 s	0.90±0.01	63±19	106±36	0.90	25	31

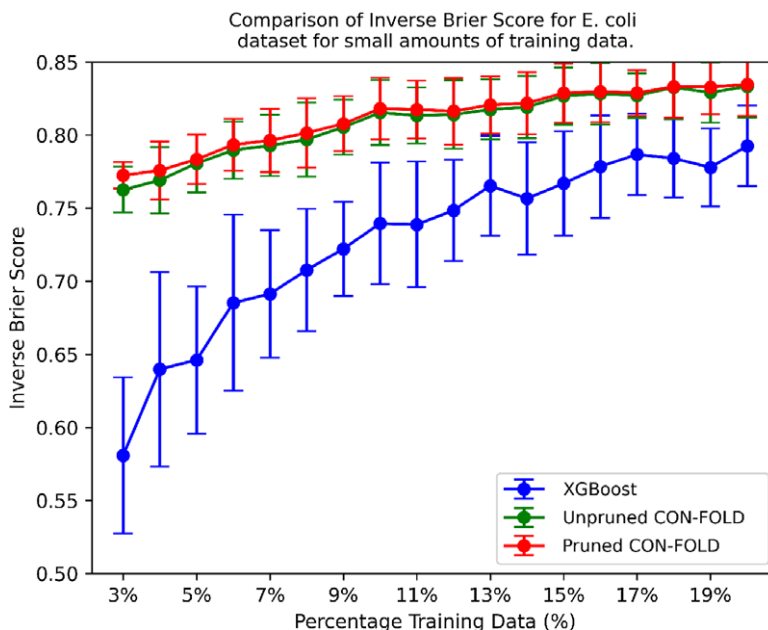


Fig. 3. Plot of IBS against the percentage of data included in the stratified training data for the *E. coli* UCI dataset. Thirty trials for each condition were performed, and error bars indicate one standard deviation across the trials. Pruned CON-FOLD used a confidence threshold of 0.65 and a pruning threshold of 0.07.

the accuracy and number of rules for XGBoost and FOLD-RM are very similar to the results given in the 2023 study and are confident that the experiments are equivalent. The times measured are wall time measured when the result is run on a PC with an Intel Core i9-13900K CPU with 64 GB of RAM. Note that when only small amounts of training data were used, we ensured that at least one example of each class was included in the training data; we will refer to this as stratified training data.

Figure 3 demonstrates how the amount of training data impacts the IBS for XGBoost and the CON-FOLD algorithm with and without pruning.

One particular use case where high levels of explainability are required is marking students' responses to exam questions. When artificial intelligence or ML is applied to automated marking, it is desirable for a minimal training dataset. This reduces the cost to mark initial examples to use as training data for the models. Therefore, we tested each model on very small training datasets with the first having just three examples of student work (0.2% of the dataset).

Three different models were tested. The first two were XGBoost and the unpruned CON-FOLD algorithm. The third model tested was the CON-FOLD algorithm with the marking scheme given as domain knowledge before training. Each rule from the marking scheme was given a confidence of 0.99. Thirty trials were conducted with randomly generated stratified data ranging from 0.2% to 90% of the data used for training. The results of this experiment can be seen in Figure 4. The average is represented by the points on the graph, and the standard deviation from the 30 trials is given as error bars.

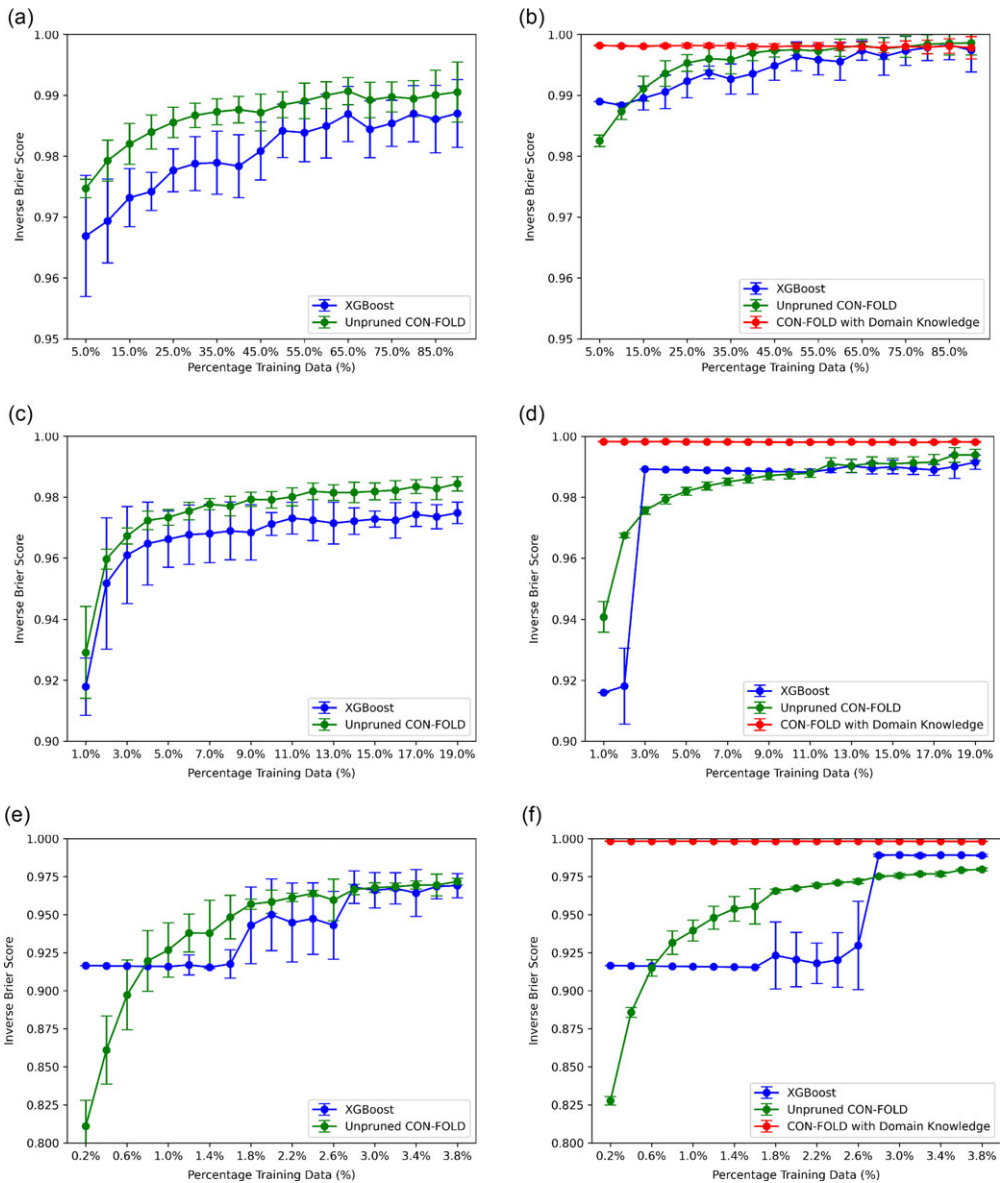


Fig. 4. Each of the plots shows the performance of models using the Inverse Brier Score metric with different amounts of training data. Plots a and b show the regimes where large amounts of training data are available, while plots e and f explore model performance with very small amounts of training data available. Plots a, c, and e use automatic feature extraction, while plots b, d, and f use manual feature extraction using regular expressions, which allows for domain knowledge in the form of a marking scheme to be included. The total number of student responses was $n = 1525$.

Plots of IBS against amounts of training data for grading student responses to an Australian Physics Olympiad problem with and without manual feature extraction.

8 Discussion

The runtime for XGBoost on the weight lifting data in Table 1 appears anomalously long. This has been observed previously (Wang and Gupta 2023) and can be attributed to the large number of features ($m = 155$) in the dataset, which is an order of magnitude greater than the others. This result confirms that both CON-FOLD and the pruning algorithm scale well with the number of input features.

Table 1 shows that the pruning algorithm reduces the number of rules compared with the FOLD-RM algorithm with a small decrease in accuracy. The main advantage of decreasing the number of rules is that it makes the results more interpretable to humans, as having hundreds of rules becomes quite difficult for a human to follow. Furthermore, a smaller set of rules reduces the inference time of the model. The FOLD-SE algorithm produces a smaller number of rules and higher accuracy for most datasets. We note that the CON-FOLD pruning technique and the use of Gini Impurity from FOLD-SE are not mutually exclusive and applying both may result in even more concise results while maintaining performance.

Our experiment in Figure 3 shows that for small amounts of stratified training data, the ability to put confidence values on predictions gives CON-FOLD a significant advantage over XGBoost. We also note that pruning gives a very slight advantage in cases of very small training datasets. We attribute this to the prevention of the model from overestimating confidence from only a small number of examples.

For the physics marking dataset, the rules created from the marking scheme align almost perfectly with the scores that students were actually awarded, as expected. This results in very strong performance even when there is very little training data. We note that the unpruned CON-FOLD algorithm gradually increases its IBS as the amount of training data increases. We attribute this to a combination of increasing confidence in rules that were learned and being able to learn more complex rules from larger training datasets.

For the XGBoost experiments with features generated by regular expressions in Figure 4f, we note that the algorithm is able to score approximately 0.92 consistently until 1.8% of the training data is reached. Then the model's performance seems to vary wildly between trials before jumping to an accuracy of 0.99 at 2.8%. We attribute the instability as sensitivity to specific training examples being included in its training data. Once 2.8% of the data is included in training, this seems to settle as this is a sufficient amount that the required examples reliably fall into the training dataset. A similar pattern is also present in Figure 4c.

For small amounts of training data, IBS of the CON-FOLD algorithm is mostly independent on whether regular expression features were included or not. However, in the regime of large amounts of training data shown in Figure 4a and Figure 4d, manually extracted features make a small but significant improvement in the performance of both XGBoost and CON-FOLD. This has implications for the use of automated feature POS tagging tools when being used for feature extraction for automated marking.

Regular expressions for feature extraction allow for more accurate results and for the implementation of background rules, but this comes at significant development costs.

9 Conclusion and future work

We have introduced confidence values that allow users to know the probability that a rule from a FOLD model will be correct when applied to a dataset. This removes the illusion of certainty when using rules created by the FOLD algorithm. We introduce a pruning algorithm that can use these confidence values to decrease the number and depth of rules. The pruning algorithm allows for the number of rules to be significantly decreased with a small impact on performance; however, it is not as effective as the use of the Gini Impurity methods in FOLD-SE.

IBS is a metric that can be used to reward accurate forecasting of probabilities of rules while maintaining compatibility with non-probabilistic models by reducing accuracy in the case of non-probabilistic predictions.

CON-FOLD allows for the inclusion of background and initial knowledge into FOLD models. We use the marking of short answer physics exams as a potential use case to demonstrate the effectiveness of incorporating readily-available domain knowledge in the form of a marking scheme. With this background knowledge, the CON-FOLD model's performance is significantly improved and out performs XGBoost, especially in the presence of small amounts of training data.

Besides improvements to the FOLD algorithm, an area for future work is feature extraction from short snippets of free-form text. NLP feature extraction tools were not able to capture the required features to implement a marking scheme, but otherwise were able to extract enough features to allow for accuracy and IBS of over 99% in the regime of large amounts of training data. OCR could allow for the grading of hand-written responses to be explored. As a final suggestion for future research, more advanced NLP tools such as large language models could be used to allow for the automated extraction of numbers and units from text.

Acknowledgments

The authors would like to thank Australian Science Innovations for access to data from the 2023 Australian Physics Olympiad. This research was supported by a scholarship from CSIRO's Data61. The ethical aspects of this research have been approved by the ANU Human Research Ethics Committee (Protocol 2023/1362). All code can be accessed on GitHub at <https://github.com/lachlanmcg123/CONFOLD>. We thank Daniel Smith for helpful comments.

References

- AGRESTI, Alan and COULL, Brent A. 1998. Approximate is better than "Exact" for interval estimation of binomial proportions. *The American Statistician* 52, 2, 119–126. doi: [10.2307/2685469](https://doi.org/10.2307/2685469).

- ALLEN, Sam, FERRO, Christopher A. T. and KWASNIOK, Frank 2023. A conditional decomposition of proper scores: quantifying the sources of information in a forecast. *Quarterly Journal of the Royal Meteorological Society* 149, 754, 1704–1725. doi: [10.1002/qj.4478](https://doi.org/10.1002/qj.4478).
- BAUMGARTNER, Peter and TARTAGLIA, Elena 2023. Bottom-up stratified probabilistic logic programming with fusemate. *EPTCS* 385, 87–100.
- BRIER, Glenn W. 1950. Verification of forecasting expressed in term of probability I. *Monthly Weather Review* 78, 1, 1–3. doi: [10.1175/1520-0493\(1950\)0780001](https://doi.org/10.1175/1520-0493(1950)0780001).
- CARENINI, Giuseppe, NG, Raymond T. and ZWART, Ed 2005. Extracting knowledge from evaluative text, Association for Computing Machinery, InProc. of the 3rd international conference on Knowledge capture, K-CAP '05, New York, NY, USA, 11–18, doi: [10.1145/1088622.1088626](https://doi.org/10.1145/1088622.1088626)
- CHEN, Tianqi and GUESTRIN, Carlos 2016. XGBoost: A Scalable Tree Boosting System, Association for Computing Machinery, Proc. of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, New York, NY, USA, 785–794, doi: [10.1145/2939672.2939785](https://doi.org/10.1145/2939672.2939785).
- DE RAEDT, Luc, KIMMIG, Angelika and TOIVONEN, Hannu January 2007. ProbLog: a probabilistic prolog and its application in link discovery. Proc. of the 20th international joint conference on Artificial intelligence, IJCAI'07, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc, 2468–2473.
- HONNIBAL, Matthew and MONTANI, Ines. 2017. SpaCy 2: natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. URL: <https://spacy.io/>. [Accessed on February 1, 2024].
- JEWSON, Stephen . 2004. *The problem with the Brier score*. arXiv:physics/0401046.
- Kaggle. 2012. *Titanic-Machine Learning from Disaster*. Kaggle. URL: <https://kaggle.com/competitions/titanic>
- KELLY, Markelle, LONGJOHN, Rachel and NOTTINGHAM, Kolby. 2024. Home - UCI Machine Learning Repository. URL: <https://archive.ics.uci.edu/>.
- LAW, Mark, RUSSO, Alessandra, and BRODA, Krysia. 2020. The ILASP system for inductive learning of answer set programs. *Theory and Practice of Logic Programming*, 20(4-5), 633–652. <https://doi.org/10.1017/S1471068420000257>
- LIU, Jing, YU, Jin, LIN, Shen, ZHANG, Guodong, ZHANG, Shuo, LI, Min and LIN, Xiaoyue. 2023. Research on rainbow probabilistic forecast model based on meteorological conditions in ZhaoSu region. *Meteorological Applications* 30, 3, e2131. doi: [10.1002/met.2131](https://doi.org/10.1002/met.2131).
- LIU, Qing, WANG, Jing, ZHANG, Dehai, YANG, Yun and WANG, NaiYao 2018. Text features extraction based on TF-IDF associating semantic. In 2018 IEEE 4th International Conference On Computer and Communications (ICCC), 2338–2343, [10.1109/CompComm.2018.8780663](https://doi.org/10.1109/CompComm.2018.8780663). URL <https://ieeexplore.ieee.org/document/8780663>
- MURPHY, Allan H. and EPSTEIN, Edward S. 1967. A note on probability forecasts and “Hedging”. *Journal of Applied Meteorology and Climatology* 6, 6, 1002–1004. doi: [10.1175/1520-0450\(1967\)006](https://doi.org/10.1175/1520-0450(1967)006), Publisher: American Meteorological Society Section: Journal of Applied Meteorology and Climatology, URL https://journals.ametsoc.org/view/journals/apme/6/6/1520-0450_1967_006_1002_anopfa_2_0_co_2.xml.
- PADALKAR, Parth, WANG, Huaduo and GUPTA, Gopal 2024. NeSyFOLD: a framework for interpretable image classification. *Proceedings of the AAAI Conference On Artificial Intelligence* 38, 5, 4378–4387. doi: [10.1609/aaai.v38i5.28235](https://doi.org/10.1609/aaai.v38i5.28235). URL <https://ojs.aaai.org/index.php/AAAI/article/view/28235>.
- QUINLAN, J. R. 1987. Simplifying decision trees. *International Journal of Man-Machine Studies* 27, 3, 221–234. doi: [10.1016/S0020-7373\(87\)80053-6](https://doi.org/10.1016/S0020-7373(87)80053-6), URL <https://www.sciencedirect.com/science/article/pii/S0020737387800536>,

- QUINLAN, J. R. 1990a. 5 - Probabilistic decision tree. In *Machine Learning*, KODRATOFF Yves and MICHALSKI Ryszard S. Eds. San Francisco (CA): Morgan Kaufmann, 140–152. doi:10.1016/B978-0-08-051055-2.50011-0, URL <https://www.sciencedirect.com/science/article/pii/B9780080510552500110>, ISBN 978-0-08-051055-2.
- QUINLAN, J. R. 1990b. Learning logical definitions from relations. *Machine Learning* 5, 3, 239–266. doi: 10.1023/A:1022699322624.
- SHAKERIN, Farhad, SALAZAR, Elmer and GUPTA, Gopal. 2017. A new algorithm to automate inductive learning of default theories. *Theory and Practice of Logic Programming* 17, 5-6, 1010–1026. doi: 10.1017/S1471068417000333. ISSN 1471-0684, 1475-3081.
- VICIENT, Carlos, SÁNCHEZ, David and MORENO, Antonio. 2013. An automatic approach for ontology-based feature extraction from heterogeneous textual resources. *Engineering Applications of Artificial Intelligence* 26, 3, 1092–1106. doi 10.1016/j.engappai.2012.08.002.
- WANG, Huaduo. 2022. Explainable AI algorithms for classification tasks with mixed data. URL: <https://utd-ir.tdl.org/items/89289f1a-c517-42bc-bb32-a7f3337a7410>
- WANG, Huaduo and GUPTA, Gopal 2022. FOLD-R++: a scalable toolset for automated inductive learning of default theories from mixed data. In *Functional and Logic Programming*. HANUS, Michael and IGARASHI, Atsushi Eds. Springer International Publishing, 224–242. doi: 10.1007/978-3-030-99461-7_13, ISBN 978-3-030-99461-7.
- WANG, Huaduo and GUPTA, Gopal 2023. FOLD-SE: an efficient rule-based machine learning algorithm with scalable explainability. In *Practical Aspects of Declarative Languages*, GEBSER, Martin and SERGEY, Ilya Eds. Springer Nature Switzerland, pp. 37–53. doi: 10.1007/978-3-031-52038-9_3.
- WANG, Huaduo, SHAKERIN, Farhad and GUPTA, Gopal 2022. FOLD-RM: a scalable, efficient, and explainable inductive learning algorithm for multi-category classification of mixed data. *Theory and Practice of Logic Programming* 22, 5, 658–677.
- WILSON, Edwin B. 1927. Probable inference, the law of succession, and statistical inference. *Journal of the American Statistical Association* 22, 158, 209–212. doi: 10.2307/2276774
- ZHENZHEN, Qian Z.Q. 2024. English sentence semantic feature extraction method based on fuzzy logic algorithm. *Journal of Electrical Systems* 20, 1, 262–275. doi: 10.52783/jes.681, URL <https://journal.esrgroups.org/jes/article/view/681>.